Refine Search

Search Results -

Terms	Documents
L68 and (compare or comparison)	1

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, March 31, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L76	l68 and (compare or comparison)	1	L76
L75	l67 and (compare or comparison)	1	L75
L74	L73 and (ssc or "source code control")	12	L74
L73	L72 and compare	15	L73
L72	(database or data with base) near (item or data) near version	56	L72
L71	(database or data with base) near (item or data) near version near compar$	0	L71
L70	5835601.uref.	22	L70
L69	5386559.uref.	34	L69
L68	5835601.pn.	2	L68
L67	5386559.pn.	2	L67
L66	717/140	359	L66
L65	717/124	457	L65

L64	717/114	284	L64
L63	717/110	158	L63
L62	717/108	388	L62
L61	717/117	78	L61
L60	717/102	64	L60
L59	717/100	419	L59
L58	717.clas.	6151	L58
L57	717/1	978	L57
L56	713/1	2226	L56
L55	713.clas.	18752	L55
L54	709/106	598	L54
L53	709/102	1289	L53
L52	709/101	667	L52
L51	709.clas.	27486	L51
L50	707.clas.	19657	L50
L49	707/203	2201	L49
L48	707/206	861	L48
L47	707/200	3139	L47
L46	707/104.1	3794	L46
L45	707/100	4343	L45
L44	707/10	8044	L44
L43	707/1	6179	L43
L42	L41 and second near program	1	L42
L41	L39 and item near check$	113	L41
L40	L39 and program near item near check$	0	L40
L39	L38 and source near code	6877	L39
L38	L37 and (item or data)	51223	L38
L37	program and system and (database or data with base) and version	52290	L37
L36	L35 and (sql or "structured query language")	193	L36
L35	L34 and version$	4936	L35
L34	L33 and (item or data)	7085	L34
L33	L32 and (database or data with base)	7714	L33
L32	("source code control" or "ssc")	16761	L32
L31	L30 and (sql or "sequential query language")	8	L31
L30	L29 and ("source code control" or "ssc")	96	L30
L29	(database or data with base) near (item or data) same version$	788	L29
L28	l25 and (database or data with base) near (item or data) same version$	13	L28
L27	L25 and (check near in or checkin)near(item or data)	0	L27
L26	L25 and check near in same (item or data)	0	L26
L25	source near control near system	1222	L25
L24	L23 and version near manag$	16	L24

L23	l21 and ("sql" or "sequential near query near language")	201	L23
L22	L21 and ("source code control system" or "ssc system")	14	L22
L21	(database or data with base) near (item or data) same version$	788	L21
L20	L19 and (sql or sequential near query near language)	43	L20
L19	L18 and source near code	54	L19
L18	L17 and version near manag$	71	L18
L17	L16 and (database or data with base)	1542	L17
L16	(smalltalk or small near talk)	2318	L16
L15	L14 and version near manag$	6	L15
L14	L11 and (smalltalk or small near talk)	8	L14
L13	L11 and (text or stream)	32	L13
L12	L11 and text or stream	807921	L12
L11	L10 and (sql or "sequential query language")	33	L11
L10	L9 and search and editor same program	254	L10
L9	717.clas.	6151	L9
L8	L7 and item	33	L8
L7	L6 and (checkin or check adj in and checkout or check adj out)	33	L7
L6	L5 and (database or data with base)	198	L6
L5	L4 and version	198	L5
L4	L3 and programs	249	L4
L3	L1 and source near code	255	L3
L2	L1 and text and binary near stream	7	L2
L1	(database or data with base) near item	2656	L1

END OF SEARCH HISTORY

Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term:

Display: 10 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ **Hit List** ◉ **Hit Count** ○ **Side by Side** ○ **Image**

Search Clear Interrupt

Search History

DATE: Wednesday, March 31, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L70	5835601.uref.	22	L70
L69	5386559.uref.	34	L69
L68	5835601.pn.	2	L68
L67	5386559.pn.	2	L67
L66	717/140	359	L66
L65	717/124	457	L65
L64	717/114	284	L64
L63	717/110	158	L63
L62	717/108	388	L62
L61	717/117	78	L61
L60	717/102	64	L60
L59	717/100	419	L59
L58	717.clas.	6151	L58
L57	717/1	978	L57
L56	713/1	2226	L56
L55	713.clas.	18752	L55

L54	709/106	598	L54
L53	709/102	1289	L53
L52	709/101	667	L52
L51	709.clas.	27486	L51
L50	707.clas.	19657	L50
L49	707/203	2201	L49
L48	707/206	861	L48
L47	707/200	3139	L47
L46	707/104.1	3794	L46
L45	707/100	4343	L45
L44	707/10	8044	L44
L43	707/1	6179	L43
L42	L41 and second near program	1	L42
L41	L39 and item near check$	113	L41
L40	L39 and program near item near check$	0	L40
L39	L38 and source near code	6877	L39
L38	L37 and (item or data)	51223	L38
L37	program and system and (database or data with base) and version	52290	L37
L36	L35 and (sql or "structured query language")	193	L36
L35	L34 and version$	4936	L35
L34	L33 and (item or data)	7085	L34
L33	L32 and (database or data with base)	7714	L33
L32	("source code control" or "ssc")	16761	L32
L31	L30 and (sql or "sequential query language")	8	L31
L30	L29 and ("source code control" or "ssc")	96	L30
L29	(database or data with base) near (item or data) same version$	788	L29
L28	l25 and (database or data with base) near (item or data) same version$	13	L28
L27	L25 and (check near in or checkin)near(item or data)	0	L27
L26	L25 and check near in same (item or data)	0	L26
L25	source near control near system	1222	L25
L24	L23 and version near manag$	16	L24
L23	l21 and ("sql" or "sequential near query near language")	201	L23
L22	L21 and ("source code control system" or "ssc system")	14	L22
L21	(database or data with base) near (item or data) same version$	788	L21
L20	L19 and (sql or sequential near query near language)	43	L20
L19	L18 and source near code	54	L19
L18	L17 and version near manag$	71	L18
L17	L16 and (database or data with base)	1542	L17
L16	(smalltalk or small near talk)	2318	L16
L15	L14 and version near manag$	6	L15
L14	L11 and (smalltalk or small near talk)	8	L14

L13	L11 and (text or stream)	32	L13
L12	L11 and text or stream	807921	L12
L11	L10 and (sql or "sequential query language")	33	L11
L10	L9 and search and editor same program	254	L10
L9	717.clas.	6151	L9
L8	L7 and item	33	L8
L7	L6 and (checkin or check adj in and checkout or check adj out)	33	L7
L6	L5 and (database or data with base)	198	L6
L5	L4 and version	198	L5
L4	L3 and programs	249	L4
L3	L1 and source near code	255	L3
L2	L1 and text and binary near stream	7	L2
L1	(database or data with base) near item	2656	L1

END OF SEARCH HISTORY

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L42: Entry 1 of 1 File: USPT Jun 2, 1998

US-PAT-NO: 5761511
DOCUMENT-IDENTIFIER: US 5761511 A

TITLE: Method and apparatus for a type-safe framework for dynamically extensible objects

DATE-ISSUED: June 2, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gibbons; Jonathan J.	Mountain View	CA		
Day; Michael J.	Mountain View	CA		
Goldstein; Theodore C.	Palo Alto	CA		
Jordan; Michael J.	Palo Alto	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Palo Alto	CA			02

APPL-NO: 08/ 187972 [PALM]
DATE FILED: January 28, 1994

INT-CL: [06] G06 F 9/45

US-CL-ISSUED: 395/705
US-CL-CURRENT: 717/122

FIELD-OF-SEARCH: 395/700, 395/164, 395/685, 395/701, 395/702, 395/705

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5315703	May 1994	Matheny et al.	395/164
☐	5339438	August 1994	Conner et al.	395/700
☐	5361350	November 1994	Conner et al.	395/600
☐	5418964	May 1995	Conner et al.	395/700
☐	5421016	May 1995	Conner et al.	395/700

☐	5428792	June 1995	Conner et al.	395/700
☐	5493680	February 1996	Danforth	395/700
☐	5502839	March 1996	Kolnick	395/800

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
A-0 361 737	September 1989	EP	
A-0 546 682	November 1992	EP	

OTHER PUBLICATIONS

IBM Technical Disclosure Bulletin, vol. 35, No. 4B, Sep. 1992, Armonk, NY, pp. 460-463.
IBM Technical Disclosure Bulletin, vol. 36, No. 6B, Jun. 1993, Armonk, NY, pp. 509-511.
Proceedings of the Object Oriented Programming Systems Languages and Applications Conference, vol. 24, 1-6 Oct. 1989, New Orleans, pp. 49-70.
Computer Journal, vol. 33, No. 3, Jun. 1990, Cambridge, GB, pp. 279-280.
"An Extensible Programming Environment for Modula-3" by Mick Jordan, ACM Proceedings SIGSOFT '90 (Irvine, CA), pp. 66-76.
Smid et al., "Two Models for the Reconstruction Problem for Dynamic Data Structures," Journal of Information Processing and Cybernetics, vol. 25 (1989) num 4, pp. 131-155.

ART-UNIT: 274

PRIMARY-EXAMINER: Trammell; James P.

ASSISTANT-EXAMINER: Corcoran, III; Peter J.

ATTY-AGENT-FIRM: Beyer & Weaver, LLP

ABSTRACT:

The present invention provides a system and process for making use of pre-existing data-structures which represent a computer program, in a way which has the advantages of shortening the time and cost required to create a new version of the computer program. The pre-existing data-structure is modified to produce a shadow data-structure which contains only shadows of those elements or nodes of the pre-existing data-structure required to perform the tasks of the new version of the computer program. The present invention includes processes to make the data-structure of the original program shadowable; processes to use data from the original program compilation process in compiling the new version of the program, including processes to create a shadow data-structure; and processes to use the new version of the computer program along with the shadow data-structure to create the desired execution. This new version of the computer program is typically a tool for checking or observing the original program's execution in some manner. Moreover, the system and processes disclosed provide mechanisms for a software manufacturer to create type-safe versions of a connected collection of objects which are dynamically extensible.

23 Claims, 21 Drawing figures

Refine Search

Search Results -

Terms	Documents
717/140	359

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, March 31, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L66	717/140	359	L66
L65	717/124	457	L65
L64	717/114	284	L64
L63	717/110	158	L63
L62	717/108	388	L62
L61	717/117	78	L61
L60	717/102	64	L60
L59	717/100	419	L59
L58	717.clas.	6151	L58
L57	717/1	978	L57
L56	713/1	2226	L56
L55	713.clas.	18752	L55
L54	709/106	598	L54
L53	709/102	1289	L53

L52	709/101	667	L52
L51	709.clas.	27486	L51
L50	707.clas.	19657	L50
L49	707/203	2201	L49
L48	707/206	861	L48
L47	707/200	3139	L47
L46	707/104.1	3794	L46
L45	707/100	4343	L45
L44	707/10	8044	L44
L43	707/1	6179	L43
L42	L41 and second near program	1	L42
L41	L39 and item near check$	113	L41
L40	L39 and program near item near check$	0	L40
L39	L38 and source near code	6877	L39
L38	L37 and (item or data)	51223	L38
L37	program and system and (database or data with base) and version	52290	L37
L36	L35 and (sql or "structured query language")	193	L36
L35	L34 and version$	4936	L35
L34	L33 and (item or data)	7085	L34
L33	L32 and (database or data with base)	7714	L33
L32	("source code control" or "ssc")	16761	L32
L31	L30 and (sql or "sequential query language")	8	L31
L30	L29 and ("source code control" or "ssc")	96	L30
L29	(database or data with base) near (item or data) same version$	788	L29
L28	l25 and (database or data with base) near (item or data) same version$	13	L28
L27	L25 and (check near in or checkin)near(item or data)	0	L27
L26	L25 and check near in same (item or data)	0	L26
L25	source near control near system	1222	L25
L24	L23 and version near manag$	16	L24
L23	l21 and ("sql" or "sequential near query near language")	201	L23
L22	L21 and ("source code control system" or "ssc system")	14	L22
L21	(database or data with base) near (item or data) same version$	788	L21
L20	L19 and (sql or sequential near query near language)	43	L20
L19	L18 and source near code	54	L19
L18	L17 and version near manag$	71	L18
L17	L16 and (database or data with base)	1542	L17
L16	(smalltalk or small near talk)	2318	L16
L15	L14 and version near manag$	6	L15
L14	L11 and (smalltalk or small near talk)	8	L14
L13	L11 and (text or stream)	32	L13
L12	L11 and text or stream	807921	L12

L11	L10 and (sql or "sequential query language")	33	L11
L10	L9 and search and editor same program	254	L10
L9	717.clas.	6151	L9
L8	L7 and item	33	L8
L7	L6 and (checkin or check adj in and checkout or check adj out)	33	L7
L6	L5 and (database or data with base)	198	L6
L5	L4 and version	198	L5
L4	L3 and programs	249	L4
L3	L1 and source near code	255	L3
L2	L1 and text and binary near stream	7	L2
L1	(database or data with base) near item	2656	L1

END OF SEARCH HISTORY

First Hit Fwd Refs

Generate Collection Print

L70: Entry 10 of 22 File: USPT Apr 2, 2002

US-PAT-NO: 6366933
DOCUMENT-IDENTIFIER: US 6366933 B1

TITLE: Method and apparatus for tracking and viewing changes on the web

DATE-ISSUED: April 2, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Ball; Thomas J.	Naperville	IL		
Douglis; Frederick	Somerset	NJ		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
AT&T Corp.	New York	NY			02

APPL-NO: 08/ 549359 [PALM]
DATE FILED: October 27, 1995

INT-CL: [07] G06 F 17/21

US-CL-ISSUED: 707/511; 707/513
US-CL-CURRENT: 715/511; 715/513

FIELD-OF-SEARCH: 395/148, 395/772, 707/511, 707/501, 707/513, 707/203, 707/202, 707/204

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4807182	February 1989	Queen	395/772
☐	4912637	March 1990	Sheedy et al.	
☐	5008853	April 1991	Bly et al.	
☐	5278979	January 1994	Foster et al.	707/203
☐	5325478	June 1994	Shelton et al.	395/772
☐	5438661	August 1995	Ogawa	395/772
☐	5535332	July 1996	Ishida	709/205

☐	5671428	September 1997	Muranaga et al.	707/511
☐	5752245	May 1998	Parrish et al.	707/10
☐	5764972	June 1998	Crouse et al.	707/204 X
☐	5806078	September 1998	Hug et al.	707/511
☐	5835601	November 1998	Shimbo et al.	707/530 X
☐	5835911	November 1998	Nakagawa et al.	707/203
☐	5860071	January 1999	Ball et al.	707/100
☐	5995097	November 1999	Tokumine et al.	707/203 X

OTHER PUBLICATIONS

Warren Ernst, "Using Netscape", QUE Corporation, pp. 34, 58-59, 66-71, 90, and 93-95, Mar. 1995.*
Using FrameMaker, Frame Technology Corp., pp. 22-1 to 22-19, Sep. 1993.*
Ball et al, "An Internet Difference Engine and its Applications", Proceedings of COMPCON '96, IEEE, pp. 71-76, Feb. 1996.*
Douglis et al, "Tracking and Viewing Changes on the Web", Proceedings of the 1996 Usenix Technical Conference, The Usenix Association, pp. 165-176, Jan. 1996.*
J-G. Lim, "Using Coollists to Index HTML Documents in the Web", Computer Networks and ISDN Systems, vol. 28, pp. 147-154, Dec. 1995.*
Pazzani et al, "Learning from Hotlists and Coldlists: Towards a WWW Information Filtering and Seeking Agent", Proceedings of the International Conference on Tools with Artificial Intelligence, pp. 492-495, Jan. 1995.*
Sheth et al, "Evolving Agents for Personalized Information Filtering", Proceedings of the Ninth Conference on Artificial Intelligence for Applications, IEEE Computer Society Press, pp. 345-352, Mar. 1993.

ART-UNIT: 2176

PRIMARY-EXAMINER: Feild; Joseph H.

ABSTRACT:

A system for accessing documents contained in a remote repository, which change in content from version-to-version. The system allows users to specify lists of documents of interest. Based on the lists, the system maintains an archive, which contains a copy of one version of each listed document, and material from which the other versions can be reconstructed. The system periodically compares the archive with current versions of the documents located in the repository, and updates the archive, thereby maintaining the ability to reconstruct current versions. The system also monitors access to the versions by each user. When a user calls for a current version, the system presents the current version, and indicates what parts of the current version have not been previously accessed by the user.

10 Claims, 18 Drawing figures

First Hit Fwd Refs

Generate Collection Print

L69: Entry 26 of 34 File: USPT Apr 8, 1997

US-PAT-NO: 5619700
DOCUMENT-IDENTIFIER: US 5619700 A

TITLE: Method and device for managing programs

DATE-ISSUED: April 8, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Abe; Yoshinari	Kawasaki			JP

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Fujitsu Limited	Kawasaki			JP	03

APPL-NO: 08/ 489951 [PALM]
DATE FILED: June 13, 1995

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
JP	6-201073	August 25, 1994

INT-CL: [06] G06 F 9/45

US-CL-ISSUED: 395/703; 395/619, 364/DIG.1, 364/261, 364/280.4, 364/283.1
US-CL-CURRENT: 717/122; 707/203

FIELD-OF-SEARCH: 395/700, 395/600

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4558413	December 1985	Schmidt et al.	364/300
☐	4912637	March 1990	Sheedy et al.	364/300
☐	5278979	January 1994	Foster et al.	395/600
☐	5357631	October 1994	Howell et al.	395/600
☐	5367683	November 1994	Brett	395/700

☐	5386559	January 1995	Eisenberg et al.	395/600
☐	5559991	September 1996	Kahfi	395/489

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
58-60366	April 1983	JP	
63-37427	February 1988	JP	
5-61666	March 1993	JP	

OTHER PUBLICATIONS

"Shared Libraries for HP-UX", Hewlett-Packard Journal, Jun. 1992, v. 43 p. 46(8). Unix System v. 386, Release 3.2,"Programmer's Guide vol. 11", AT&T, Prentice-Hall, 1989, pp. 14-1 14-11.

ART-UNIT: 236

PRIMARY-EXAMINER: Kriess; Kevin A.

ASSISTANT-EXAMINER: Chaki; Kakali

ATTY-AGENT-FIRM: Staas & Halsey

ABSTRACT:

A file storage section stores source files without duplication. A management information storage section stores management information for managing the relationship between each program version and source files used by it and the sharing relationship between each source file and other program versions that share it. When a source file is read and then edited, a storage method determining section determines whether or not the version series of the source file is to be divided. When the source file is to be stored without dividing its version series, a storage processing section stores the results of edition in the file storage section as an update version of the same series as the original source file. When it is to be stored with its version series divided, the version series is divided and the source file is stored as a new version series. A management information control section updates the management information when the source file is stored.

17 Claims, 4 Drawing figures

L24: Entry 15 of 16 File: USPT Aug 29, 2000

DOCUMENT-IDENTIFIER: US 6112024 A
TITLE: Development system providing methods for managing different versions of objects with a meta model

Brief Summary Text (11):
An additional problem with present-day systems is that of lack of scalability. Existing file-based systems do not provide scalability methodology, such as parallel processing. Unlike SQL database systems, for instance, file-based systems do not include an architectural design which support; a high transaction volume--a high number of users accessing a particular object or set of objects. All told, since existing products are all based on file-based technology, they are ill-suited to scale to a level supporting 1,000 or more users at a time.

Detailed Description Text (7):
Also shown, the software system 150 includes an integrated development tool front-end 170 having an "Object Cycle" versioning System 175 of the present invention. The RDBMS front-end 170 may comprise any one of a number of database development front-ends, including Powersoft PowerBuilder.TM., Powersoft Optima++.TM., dBASE.RTM., Paradox.RTM., Microsoft.RTM. Access, Microsoft.RTM. Visual C++, or the like. In an exemplary embodiment, the front-end will also include SQL access drivers (e.g., Borland SQL Links, or Microsoft ODBC drivers) for accessing SQL database server tables in a Client/Server environment. The Object Cycle versioning system itself will now be described in greater detail.

Detailed Description Text (36):
The next layer in this multi-layer architecture is the primary layer 830. The primary layer 830 forms the core or workhorse layer for the engine; it performs the bulk of the work of version management. Depending on the particular API call being processed, the primary layer 830 may work in conjunction with lower level layer 840 or, alternatively, interact directly with the database primary layer 850. The lower level layer 840, on the other hand, typically includes methods which require the system to access data from the database 870. The routines of the primary layer 830 include ones which do not require data access. These routines manage status information stored in a hash table in a cache memory 835. The cache 835, which operates under the control of a cache manager, stores names status and other information (e.g., name space information), thereby affording quick access to such information in the system. Changes which affect this information are propagated to the cache 835, so that at all times the system maintains a consistent view.

Detailed Description Text (37):
The database primary layer 850 is invoked by both the primary layer 830 (for certain calls) and the lower level layer 840. The database primary layer 850 is responsible for preparing appropriate SQL statements for processing information from the database 870. Actual execution of each SOL statement is the responsibility of the database integrity layer 860.

Detailed Description Text (72):
After declaring local variables at lines 15-18, the method initializes an identifier or ID for the entity record, at line 20. Now, the method may undertake preparation of an appropriate SQL statement. At lines 24-31, the method constructs

a string (valuesBuj) which stores the entity ID, format, type, policy, and owner ID for the entity to be added. Next, the method constructs a command string (cmdBuf) which adds to the above-constructed string a SQL command (here, "INSERT"), together with the name of the table to insert into (here, "ENTITIES.sub.--TABLE").

Detailed Description Text (73):
After the SQL statement has been constructed, the method is now ready to submit the statement for execution. This is shown at lines 38-73. Specifically, the SQL statement (command buffer or cmdBuf) is passed, together with a (pointer to) connection descriptor, to the dsExecuteCmd routine, as shown at lines 41-48. This call performs the actual insertion of the entities record. Upon completion of the call, a return or result code (rc) is returned, for reporting the success of the operation. Error processing (e.g., attempting to insert a duplicate key value) is handled at lines 50-70. The method will generate a new entity ID for each repeat attempt. After successful completion of insertion of the record or after reaching a "retry limit," the method proceeds to line 75 to return the result code. Here, the method returns to the caller, the cpAddObject method. It, in turn, will return to its caller and so forth and so on until control is; ultimately returned to the ciAddObject method. After performing housekeeping and cleanup steps, the ciAddObject method concludes by returning a result code to its caller, for indicating the

Detailed Description Text (76):
A request by a client to check out an object is first received by the communication layer 810, shown previously in FIG. 8. It, in turn, passes the request to the change control manager or module 825, for changing control. The change control manager 825 validates the arguments and then passes the request to the primary layer 830. After examining entity information about the object from cache 835, the primary layer 830 will invoke the database primary layer 850 for preparing an SQL statement which requests the object. The prepared SQL statement is, in turn, executed by the database integrity layer 860. This selects the appropriate object from the database table 870. As shown in FIG. 12, this operation--"Checkout Object" method 1200--may be implemented as a specific sequence of method calls for checking out an object: ciCheckOutObject 1210 calls ciCheckOutObject 1220 which, in turn, calls cpCheckOutObject 1230 which, in turn, calls ciCheckOutBlob 1240 which, in turn, calls ciReserveCurrent 1250. Implementation of these methods will now be described in further detail.

Detailed Description Text (87):
This is a workhorse routine or method which performs most of the work in reserving and retrieving a blob. After initializing local variables at lines 14-17, the method "grabs" a connection at lines 30-32. If a connection cannot be successfully grabbed, the method returns an error code at line 34. Otherwise, the method proceeds to line 49 to validate the access policy. At this point, the method calls into the database primary layer, by invoking dbpGetEntityInfo, at lines 49-53. The entity information is returned to the address passed as the fourth parameter, (address of) entityInfo. The call to dbpGetEntityInfo, since it is a call into a database primary layer, is ultimately executed as an SQL statement against the database.

Detailed Description Text (102):
ObjectCycle.TM. is a next generation, client/server object management facility for software version control and deployment. ObjectCycle is specifically targeted toward the platforms and network services available in a typical Microsoft Windows network and can run on any machine in such a network, providing services to multiple client applications at a time. ObjectCycle manages client/server communication through the Microsoft RPC services of a Microsoft Windows network, and stores its data in a RDBMS such as Sybase SQL Anywhere 5.0 via ODBC.

Detailed Description Text (104):
The ObjectCycle Server uses the ODBC database access standard to communicate with the RDBMS. The RDBMS is used to reliably store and quickly access ObjectCycle objects. Sybase SQL, Anywhere 5.0 is a desktop RDBMS and is included with ObjectCycle. ObjectCycle Manager: The ObjectCycle Manager is an easy-to-use, graphical tool that comes in a 16 bit format for Windows 3.11 or a 32-bit format for Windows 95 and NT. ObjectCycle Manager is intended to be used by both ObjectCycle users and administrators to manage and manipulate data within the ObjectCycle Server.

Detailed Description Text (107):
ObjectCycle is a network-based client/server facility. The server component is ObjectCycle Server; the client component is ObjectCycle Manager, or other clients such as PowerBuilder. Client/Server communications are managed by Microsoft RPC. ObjectCycle Server, which developers access from their own machines, stores and tracks the development history of objects. The data is stored in the Sybase SQL Anywhere 5.0 relational database that comes with the ObjectCycle Server.

Detailed Description Paragraph Table (11):

```
                                                                                1: /*
2: * dbpAddEntity() 3: * 4: * Add an entry in the entity table 5: */ 6: CAMUSAPI
dbpAddEntity( PCCB pccb 7: , PDSCONN pDSConn 8: , cFORMAT format 9: , cOBJECTTYPE
type 10: , cACCESSPOLICY policy 11: , CAMUSID ownerID 12: , CAMUSID *pEntityID
13: ) 14: { 15: CHAR cmdBuf[ CMDBUFLEN ], 16: valuesBuf[ VALUESBUFLEN ]; 17: LONG
dupkeyRetries = 0; 18: CAMUS.sub.-- RC rc; 19: 20: *pEntityID = newCamusID(); 21:
22: // Prepare SQL statement 23: 24: sprintf( valuesBuf 25: , "%ld, %d, %d, %d, %
ld" 26: , *pEntityID 27: , format 28: , type 29: , policy 30: , ownerID 31: ) ; 32:
sprintf( cmdBuf 33: , INSERT.sub.-- CMD 34: , ENTITIES.sub.-- TABLE 35: , valuesBuf
36: ) ; 37: 38: while( dupkeyRetries++ < DUPKEY.sub.-- RETRY.sub.-- LIMIT ) 39:
{ 40: // Execute SQL Statement 41: rc = dsExecuteCmd( pccb 42: , pDSConn 43: ,
cmdBuf 44: , NULL 45: , NULL 46: , NULL 47: , NULL 48: ) ; 49: 50: if ( GET.sub.--
ERROR.sub.-- CODE( rc ) == ERR.sub.-- DS.sub.-- DUPKEY ) 51: { 52: CAMUS.sub.--
TRACE("Duplicate key hit; retrying..."); 53: CAMUS.sub.-- ASSERT( CAMUS.sub.--
SUCCESS == camErrorClear( pccb, rc )); 54: (*pEntityID) += (CAMUSID) rand(); 55:
sprintf( valuesBuf 56: , "%ld, %d, %d, %d, %ld" 57: , *pEntityID 58: , format 59: ,
type 60: , policy 61: , ownerID 62: ) ; 63: sprintf( cmdBuf 64: , INSERT.sub.-- CMD
65: , ENTITIES.sub.-- TABLE 66: , valuesBuf 67: ) ; 68: 69: continue; 70: } 71: 72:
break; 73: } 74: 75: return( rc ); 76: }
```

CLAIMS:

1. In a development system for creating programs from objects, an improved method for managing versions of the objects, the method comprising:

providing a meta model for presenting to a user the objects and versions thereof as a hierarchical representation;

for each object created, performing substeps of:

creating in said hierarchical representation a single entity node for representing the object, and

creating in said hierarchical representation at least one name node for representing one or more names for the object; and

for each version created for each corresponding object, creating in said hierarchical representation an instance node for representing said each version for each object, wherein said hierarchical representation conveys to the user a relationship between each version and its corresponding object and conveys any

semantic relationships existing between said objects.

L24: Entry 15 of 16 File: USPT Aug 29, 2000

DOCUMENT-IDENTIFIER: US 6112024 A
TITLE: Development system providing methods for managing different versions of objects with a meta model

Brief Summary Text (11):
An additional problem with present-day systems is that of lack of scalability. Existing file-based systems do not provide scalability methodology, such as parallel processing. Unlike SQL database systems, for instance, file-based systems do not include an architectural design which support; a high transaction volume--a high number of users accessing a particular object or set of objects. All told, since existing products are all based on file-based technology, they are ill-suited to scale to a level supporting 1,000 or more users at a time.

Detailed Description Text (7):
Also shown, the software system 150 includes an integrated development tool front-end 170 having an "Object Cycle" versioning System 175 of the present invention. The RDBMS front-end 170 may comprise any one of a number of database development front-ends, including Powersoft PowerBuilder.TM., Powersoft Optima++.TM., dBASE.RTM., Paradox.RTM., Microsoft.RTM. Access, Microsoft.RTM. Visual C++, or the like. In an exemplary embodiment, the front-end will also include SQL access drivers (e.g., Borland SQL Links, or Microsoft ODBC drivers) for accessing SQL database server tables in a Client/Server environment. The Object Cycle versioning system itself will now be described in greater detail.

Detailed Description Text (36):
The next layer in this multi-layer architecture is the primary layer 830. The primary layer 830 forms the core or workhorse layer for the engine; it performs the bulk of the work of version management. Depending on the particular API call being processed, the primary layer 830 may work in conjunction with lower level layer 840 or, alternatively, interact directly with the database primary layer 850. The lower level layer 840, on the other hand, typically includes methods which require the system to access data from the database 870. The routines of the primary layer 830 include ones which do not require data access. These routines manage status information stored in a hash table in a cache memory 835. The cache 835, which operates under the control of a cache manager, stores names status and other information (e.g., name space information), thereby affording quick access to such information in the system. Changes which affect this information are propagated to the cache 835, so that at all times the system maintains a consistent view.

Detailed Description Text (37):
The database primary layer 850 is invoked by both the primary layer 830 (for certain calls) and the lower level layer 840. The database primary layer 850 is responsible for preparing appropriate SQL statements for processing information from the database 870. Actual execution of each SOL statement is the responsibility of the database integrity layer 860.

Detailed Description Text (72):
After declaring local variables at lines 15-18, the method initializes an identifier or ID for the entity record, at line 20. Now, the method may undertake preparation of an appropriate SQL statement. At lines 24-31, the method constructs

a string (valuesBuj) which stores the entity ID, format, type, policy, and owner ID for the entity to be added. Next, the method constructs a command string (cmdBuf) which adds to the above-constructed string a SQL command (here, "INSERT"), together with the name of the table to insert into (here, "ENTITIES.sub.--TABLE").

Detailed Description Text (73):
After the SQL statement has been constructed, the method is now ready to submit the statement for execution. This is shown at lines 38-73. Specifically, the SQL statement (command buffer or cmdBuf) is passed, together with a (pointer to) connection descriptor, to the dsExecuteCmd routine, as shown at lines 41-48. This call performs the actual insertion of the entities record. Upon completion of the call, a return or result code (rc) is returned, for reporting the success of the operation. Error processing (e.g., attempting to insert a duplicate key value) is handled at lines 50-70. The method will generate a new entity ID for each repeat attempt. After successful completion of insertion of the record or after reaching a "retry limit," the method proceeds to line 75 to return the result code. Here, the method returns to the caller, the cpAddObject method. It, in turn, will return to its caller and so forth and so on until control is; ultimately returned to the ciAddObject method. After performing housekeeping and cleanup steps, the ciAddObject method concludes by returning a result code to its caller, for indicating the

Detailed Description Text (76):
A request by a client to check out an object is first received by the communication layer 810, shown previously in FIG. 8. It, in turn, passes the request to the change control manager or module 825, for changing control. The change control manager 825 validates the arguments and then passes the request to the primary layer 830. After examining entity information about the object from cache 835, the primary layer 830 will invoke the database primary layer 850 for preparing an SQL statement which requests the object. The prepared SQL statement is, in turn, executed by the database integrity layer 860. This selects the appropriate object from the database table 870. As shown in FIG. 12, this operation--"Checkout Object" method 1200--may be implemented as a specific sequence of method calls for checking out an object: ciCheckOutObject 1210 calls ciCheckOutObject 1220 which, in turn, calls cpCheckOutObject 1230 which, in turn, calls ciCheckOutBlob 1240 which, in turn, calls ciReserveCurrent 1250. Implementation of these methods will now be described in further detail.

Detailed Description Text (87):
This is a workhorse routine or method which performs most of the work in reserving and retrieving a blob. After initializing local variables at lines 14-17, the method "grabs" a connection at lines 30-32. If a connection cannot be successfully grabbed, the method returns an error code at line 34. Otherwise, the method proceeds to line 49 to validate the access policy. At this point, the method calls into the database primary layer, by invoking dbpGetEntityInfo, at lines 49-53. The entity information is returned to the address passed as the fourth parameter, (address of) entityInfo. The call to dbpGetEntityInfo, since it is a call into a database primary layer, is ultimately executed as an SQL statement against the database.

Detailed Description Text (102):
ObjectCycle.TM. is a next generation, client/server object management facility for software version control and deployment. ObjectCycle is specifically targeted toward the platforms and network services available in a typical Microsoft Windows network and can run on any machine in such a network, providing services to multiple client applications at a time. ObjectCycle manages client/server communication through the Microsoft RPC services of a Microsoft Windows network, and stores its data in a RDBMS such as Sybase SQL Anywhere 5.0 via ODBC.

Detailed Description Text (104):
The ObjectCycle Server uses the ODBC database access standard to communicate with the RDBMS. The RDBMS is used to reliably store and quickly access ObjectCycle objects. Sybase SQL, Anywhere 5.0 is a desktop RDBMS and is included with ObjectCycle. ObjectCycle Manager: The ObjectCycle Manager is an easy-to-use, graphical tool that comes in a 16 bit format for Windows 3.11 or a 32-bit format for Windows 95 and NT. ObjectCycle Manager is intended to be used by both ObjectCycle users and administrators to manage and manipulate data within the ObjectCycle Server.

Detailed Description Text (107):
ObjectCycle is a network-based client/server facility. The server component is ObjectCycle Server; the client component is ObjectCycle Manager, or other clients such as PowerBuilder. Client/Server communications are managed by Microsoft RPC. ObjectCycle Server, which developers access from their own machines, stores and tracks the development history of objects. The data is stored in the Sybase SQL Anywhere 5.0 relational database that comes with the ObjectCycle Server.

Detailed Description Paragraph Table (11):

```
                                                                    1: /*
2: * dbpAddEntity() 3: * 4: * Add an entry in the entity table 5: */ 6: CAMUSAPI
dbpAddEntity( PCCB pccb 7: , PDSCONN pDSConn 8: , cFORMAT format 9: , cOBJECTTYPE
type 10: , cACCESSPOLICY policy 11: , CAMUSID ownerID 12: , CAMUSID *pEntityID
13: ) 14: { 15: CHAR cmdBuf[ CMDBUFLEN ], 16: valuesBuf[ VALUESBUFLEN ]; 17: LONG
dupkeyRetries = 0; 18: CAMUS.sub.-- RC rc; 19: 20: *pEntityID = newCamusID(); 21:
22: // Prepare SQL statement 23: 24: sprintf( valuesBuf 25: , "%ld, %d, %d, %d, %
ld" 26: , *pEntityID 27: , format 28: , type 29: , policy 30: , ownerID 31: ) ; 32:
sprintf( cmdBuf 33: , INSERT.sub.-- CMD 34: , ENTITIES.sub.-- TABLE 35: , valuesBuf
36: ) ; 37: 38: while( dupkeyRetries++ < DUPKEY.sub.-- RETRY.sub.-- LIMIT ) 39:
{ 40: // Execute SQL Statement 41: rc = dsExecuteCmd( pccb 42: , pDSConn 43: ,
cmdBuf 44: , NULL 45: , NULL 46: , NULL 47: , NULL 48: ) ; 49: 50: if ( GET.sub.--
ERROR.sub.-- CODE( rc ) == ERR.sub.-- DS.sub.-- DUPKEY ) 51: { 52: CAMUS.sub.--
TRACE("Duplicate key hit; retrying..."); 53: CAMUS.sub.-- ASSERT( CAMUS.sub.--
SUCCESS == camErrorClear( pccb, rc )); 54: (*pEntityID) += (CAMUSID) rand(); 55:
sprintf( valuesBuf 56: , "%ld, %d, %d, %d, %ld" 57: , *pEntityID 58: , format 59: ,
type 60: , policy 61: , ownerID 62: ) ; 63: sprintf( cmdBuf 64: , INSERT.sub.-- CMD
65: , ENTITIES.sub.-- TABLE 66: , valuesBuf 67: ) ; 68: 69: continue; 70: } 71: 72:
break; 73: } 74: 75: return( rc ); 76: }
```

CLAIMS:

1. In a development system for creating programs from objects, an improved method for managing versions of the objects, the method comprising:

providing a meta model for presenting to a user the objects and versions thereof as a hierarchical representation;

for each object created, performing substeps of:

creating in said hierarchical representation a single entity node for representing the object, and

creating in said hierarchical representation at least one name node for representing one or more names for the object; and

for each version created for each corresponding object, creating in said hierarchical representation an instance node for representing said each version for each object, wherein said hierarchical representation conveys to the user a relationship between each version and its corresponding object and conveys any

semantic relationships existing between said objects.

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L24: Entry 16 of 16 File: USPT May 31, 1994

DOCUMENT-IDENTIFIER: US 5317731 A
TITLE: Intelligent page store for concurrent and consistent access to a database by a transaction processor and a query processor

Brief Summary Text (5):
Typically enterprises create and maintain their databases through a high volume of relatively simple transactions. Each transaction represents a well-understood business operation (creating a new customer record, noting an account payment or transfer). Increasingly enterprises are becoming interested in running more ad hoc unstructured queries against their online data. This is stimulated by the feasibility of writing these more complex queries in SQL. Typical applications might be: testing new market opportunities, decision support, detecting historical trends, profitability analysis etc.. These unstructured queries are characterized by:

Detailed Description Text (21):
The Intelligent Page Store contains a Processing part 11 and a Non-Volatile Storage part 12. The Processing in the Intelligent Page Store consists of a Version Manager 13, which handles the page read and write requests from the Transaction and Query Processors via Interfaces 5,7,18. The Non-Volatile Storage in the Intelligent Page Store acts as a repository for the Transaction Database Log 6 and the Transaction Database Data 8. This Transaction Database Data is exactly that shown as 8 in FIG. 1, i.e. it is the backing storage for the current copy of every page which the Transaction Processor 6 has written to Non-Volatile Storage. The Intelligent Page Store provides additional Non-Volatile Storage for pages of Query Version Data 14. These allow a consistent query view of the database to be presented to the Query Processor via requests in Interface 18.

Detailed Description Text (22):
The function of the Version Manager 13 is to control access to shared logical pages of data by the Transaction Processor and Query Processor while preventing them from affecting each other's performance significantly, and minimizing total requirements for physical non-volatile storage. As a result the total non-volatile storage needed to save the Transaction Database Data 8 and the Query Version Data 14 is considerably less than twice the amount which would have been needed for Transaction Database Data in the prior art.

Detailed Description Text (23):
The Version Manager 13, responds to page read requests from the Query Processor so as to present a recent consistent version of database data via Interface 18, and at the same time to appear as simple non-volatile storage in response to requests from the Transaction Processor via Interfaces 5 and 7.

Detailed Description Text (26):
To implement implicit versioning, the Version Manager 13 is responsible for routing page accesses from both transaction and query processing to the correct physical pages, for maintaining and managing versions of pages, for initiating and processing the creation of new query snapshots and for recovering and reusing

physical storage from old page versions.

Detailed Description Text (27):
The Transaction Database Log 6 saved on Non-Volatile Storage 12 in the Intelligent Page Store 10, is exactly the information which the Transaction Processor 3 needs to save on non-volatile storage to protect database data against transaction aborts and system failures. The Version Manager 13 in the Intelligent Page Store responds to read and write requests in the log Interface 5 saving the information in the Transaction Database Log 6 on the Intelligent Page Store Non-Volatile Storage and returning it to subsequent read requests. The advantage of having the Database Log in the Intelligent Page Store is that the log information can be used to create consistent versions of database data without disturbing the Transaction Processor or reducing its throughput. Since a log is maintained only for update transactions and the Query Processor needs no access to transaction log information, the amount of non-volatile storage required to store the Database Log in the Page Store is the same as if the log were directly attached to the Transaction Processor as in prior art.

Detailed Description Text (28):
Similarly the Version Manager responds to Transaction Database Data page read and write requests in Interface 7 by saving page images in the Intelligent Page Store Non-Volatile Storage for Transaction Database Data 8 and returning them on subsequent read requests. This enables the Version Manager to create an additional physical copy of pages in Query Version Data 14 to support a consistent view of the database for queries only if some transaction has modified the page since the query version was created by the database snapshot processing. Since non-volatile storage is a significant component of the cost of database installations, avoiding unnecessary separate physical copies of pages for the Transaction and Query Processors is important. Implicit versioning, described in FIG. 3, and FIG. 4 includes an efficient scheme for determining when a copy of a database data page must be made to meet the requirement of presenting a consistent view to queries and the appearance of a non-volatile medium to transaction processing. This enables queries and transactions to execute concurrently without unnecessary replication of the database pages and hence at minimal cost.

Detailed Description Text (29):
The Non-Volatile Storage 12 in the Intelligent Page Store 10 can be implemented with any standard non-volatile medium (such as magnetic disks) for storing the Transaction Database Log 6, Transaction Database Data 8 and Query Version Data 14.

Detailed Description Text (36):
The next section of FIG. 3 including 34, 35, 36, 37 shows Query Versions being created by taking a snapshot of database state and maintained for use by queries. In action 34 the Version Manager takes a snapshot of the Initial Database State 26 to create Query Version V0 whose lifetime is shown by 36. The length of 36 shows exactly the lifetime during which query version V0 is available for use by queries running on the Query Processor. Action 35 shows the Version Manager at some time after transaction X3 has committed, taking a snapshot of the consistent database state 29 and using it to create Query Version V1. The lifetime during which Query Version V1 is available to queries is shown by the exact length of 37.

Detailed Description Text (38):
The implicit versioning scheme implemented by the Version Manager 11 has to deal with the fact, that the transaction processor 3 will write out pages of data via Interface 7 in a way which optimizes the use of its Buffer 4. In particular pages will be written out including uncommitted data and without any guarantee that the set of pages written out represent a consistent state of the transaction database in the sense of database States 26, 27, 28, 29, 30, 31, 32, 33. The Intelligent Page Store, has to receive these pages and return them on subsequent read requests without disturbing the view of data seen by longer running queries in the Query

Processor; these must see the Query Versions 36, 37. In implicit versioning, this is achieved by advancing the version of data seen by the queries in discrete time steps. At each time stem a new query snapshot is created and made visible to queries. Each snapshot is a consistent view of committed database data at some recent time. Between time steps, the Intelligent Page Store presents a constant view of the data to queries. When the Transaction Processor writes out updated pages to the Intelligent Page Store, the updated pages are saved but not made visible to the queries until the next time step.

Detailed Description Text (40):
Implicit versioning also determines accurately when an additional copy of a database data page is required to support the current transactional and query views. This enables non-volatile storage requirements to be minimized. The management of page copies by implicit versioning is described by a time state diagram in FIG. 4. Time advances from left to right in this diagram. Events directly above each other are simultaneous.

Detailed Description Text (41):
The initial state of the database data i.e. the combined Transaction Database Data 8 and the Query Version Data 14, in the Intelligent Page Store 10 is shown as 47. This is actually a composite state represented by a set of logical pages P1, P2, . . . P7 which store a set of page values. We represent the values stored in these pages by letters "a", "b", "c", "d", "e", "f", "g" respectively. This database state is assumed to be a consistent state of the transaction database and the actual state of transaction data, as would occur if transaction processing had just restarted after being quiesced. State 26 in in FIG. 3 illustrated this case. We assume that a snapshot of this consistent state has been taken as shown in Action 34 and a Query Version created corresponding to it as in "Query Version VO" 36.

Detailed Description Text (42):
A key point is that in this State 47 no page is stored twice; a single copy of each of pages P1, P2, . . . P7 is adequate to support correct transaction and query views of the database data. This set of physical pages is acting as Transaction Database Data 8; no additional physical storage is required for Query Version Data 14 at this time.

Detailed Description Text (45):
The next Action 43 is a request from the Transaction Processor to write the value "x" into page P3. State 49 shows that a copy of page P3 is made so that both the old value "c" and the new value "x" can be saved. At this point queries will see the old value "c" residing in Query Version Data 14, whereas Transaction Processor requests to non-volatile storage will see the new value "x" in a copy of the page in Transaction Database Data.

Detailed Description Text (52):
Logic in Version Manager implementing Implicit Versioning

Detailed Description Text (53):
FIG. 5 shows the logic in the Version Manager 13 used to implement implicit versioning. It is a control flow graph.

Detailed Description Text (54):
The data structures in the Version Manager consist of:

Detailed Description Text (55):
page to file map This maps page numbers to the location in the file system where the primary copy of that logical page is stored. At times when the transaction database state is identical to the current Query Version (e.g. state 47 in FIG. 4) the Transaction Database Data 8 will consist of the primary copy of each logical page in the database.

Detailed Description Text (58):
Referring now to the flowchart in FIG. 5. The Input Events 55 to the Version Manager are a "request to read a page" 56, or "write a page" 57, from the Transaction Processor, to "read a page from the query processor" 58 or to "create a new query version" 59.

First Hit

End of Result Set

☐ Generate Collection Print

L31: Entry 8 of 8 File: DWPI Jul 27, 2000

DERWENT-ACC-NO: 2000-686463
DERWENT-WEEK: 200067

TITLE: Database item versioning system for hand-held devices, has source code control system to store versions of item and mechanism to check in and check out the item

INVENTOR: DENGLER, P; KRUY, S ; PAN, Z ; RAMOS, B

PATENT-ASSIGNEE: MICROSOFT CORP (MICRN)

PRIORITY-DATA: 1999US-0235038 (January 21, 1999)

Search Selected Search ALL Clear

PATENT-FAMILY:

	PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
☐	WO 200043916 A2	July 27, 2000	E	024	G06F017/30
☐	AU 200034721 A	August 7, 2000		000	G06F017/30

DESIGNATED-STATES: AE AL AM AT AU AZ BA BB BG BR BY CA CH CN CR CU CZ DE DK DM EE ES FI GB GD GE GH GM HR HU ID IL IN IS JP KE KG KP KR KZ LC LK LR LS LT LU LV MA MD MG MK MN MW MX NO NZ PL PT RO RU SD SE SG SI SK SL TJ TM TR TT TZ UA UG UZ VN YU ZA ZW AT BE CH CY DE DK EA ES FI FR GB GH GM GR IE IT KE LS LU MC MW NL OA PT SD SE SL SZ TZ UG ZW

APPLICATION-DATA:

PUB-NO	APPL-DATE	APPL-NO	DESCRIPTOR
WO 200043916A2	January 21, 2000	2000WO-US01472	
AU 200034721A	January 12, 2000	2000AU-0034721	
AU 200034721A		WO 200043916	Based on

INT-CL (IPC): G06 F 17/30

ABSTRACTED-PUB-NO: WO 200043916A
BASIC-ABSTRACT:

NOVELTY - An editor program is used to manipulate the item contained in a structured query language (SQL) database. Source code control (SCC) system stores versions of the item which are checked in and checked out by a mechanism.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the following:

(a) computer implemented method to check on item from SCC system;

(b) computer implemented method to check on item into SCC system;

(c) computer readable medium

USE - For handheld device, multiprocessor system, microscope based or programmable consumer electronics network PCs, minicomputers, mainframe computers.

ADVANTAGE - When SQL procedure is stored in the database it does not have to be replicated in each client. This saves programming effort especially when different client user interface and development systems are used. The mechanism keeps track of users who wish to edit the stored procedures and once they have been changed, the mechanism keeps track of the change that have been made to the stored procedures. Thus when a bug or error has been introduced into a stored procedure, the database administrator is able to determine the history of changes made to a particular stored procedure.

DESCRIPTION OF DRAWING(S) - The figure shows the flowchart of check out method.

ABSTRACTED-PUB-NO: WO 200043916A
EQUIVALENT-ABSTRACTS:

CHOSEN-DRAWING: Dwg.3/4

DERWENT-CLASS: T01
EPI-CODES: T01-J05B3; T01-J05B4A; T01-M06A1A;

L76: Entry 1 of 1 File: USPT Nov 10, 1998

DOCUMENT-IDENTIFIER: US 5835601 A
TITLE: File editing system and shared file editing system with file content secrecy, file version management, and asynchronous editing

Detailed Description Text (37):
Next, the operation of each element in this asynchronous editing system shown in FIG. 5 will be described for specific record management schemes. In any case, the editing procedure generation unit 501 carries out the operation corresponding to the "diff" command of the UNIX. Namely, the editing procedure generation unit 501 compares the file data in the editing target version and the file data after the editing, and outputs the editing procedure data such as "what is inserted where" and "from where to where is deleted".

Detailed Description Text (111):
Here, the judgment as to whether it can be merged or not can be made by attaching a version number of the target data to a header of the record management information to be transmitted at the client 91 side, such that the server 90 side can compare the version number in the header with the version number of the latest version stored therein, and judge that it can be merged when they coincide, and that it cannot be merged otherwise. In addition, the record data to be transmitted to the client 91 when it cannot be merged can be the record data between the latest version and the version indicated by the version number in the header of the record management information for example. At the client 91 side, the prescribed editing procedure conversion is carried out to generate a new record management information, and this is sent to the server 90 again, such that the server 90 side judges whether it can be merged or not again.

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 5835601 A

Using default format because multiple data bases are involved.

L76: Entry 1 of 1 File: USPT Nov 10, 1998

US-PAT-NO: 5835601
DOCUMENT-IDENTIFIER: US 5835601 A

TITLE: File editing system and shared file editing system with file content secrecy, file version management, and asynchronous editing

DATE-ISSUED: November 10, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Shimbo; Atsushi	Chiba-ken			JP
Takahashi; Toshinari	Tokyo			JP
Tomoda; Ichiro	Tokyo			JP
Murota; Masao	Kanagawa-ken			JP

US-CL-CURRENT: 713/165; 380/29, 707/203, 715/530

Full | Title | Citation | Front | Review | Classification | Date | Reference | Sequences | Attachments | Claims | KWIC | Draw De

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Terms	Documents
L68 and (compare or comparison)	1

Display Format: - Change Format

Previous Page Next Page Go to Doc#